Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture

Seongil O[†] Young Hoon Son[†] Nam Sung Kim[‡] Jung Ho Ahn[†]
[†]Seoul National University [‡]University of Wisconsin-Madison
{swdfish,yhson96,gajh}@snu.ac.kr nskim3@wisc.edu

Abstract

Modern DRAM devices for the main memory are structured to have multiple banks to satisfy ever-increasing throughput, energy-efficiency, and capacity demands. Due to tight cost constraints, only one row can be buffered (opened) per bank and actively service requests at a time, while the row must be deactivated (closed) before a new row is stored into the row buffers. Hasty deactivation unnecessarily re-opens rows for otherwise row-buffer hits while hindsight accompanies the deactivation process on the critical path of accessing data for row-buffer misses. The time to (de)activate a row is comparable to the time to read an open row while applications are often sensitive to DRAM latency. Hence, it is critical to make the right decision on when to close a row. However, the increasing number of banks per DRAM device over generations reduces the number of requests per bank. This forces a memory controller to frequently predict when to close a row due to a lack of information on future requests, while the dynamic nature of memory access patterns limits the prediction accuracy.

In this paper, we propose a novel DRAM microarchitecture that can eliminate the need for any prediction. First, we identify that precharging the bitlines dominates the deactivate time, while sense amplifiers that work as a row buffer are physically coupled with the bitlines such that a single command precharges both bitlines and sense amplifiers simultaneously. By decoupling the bitlines from the row buffers using isolation transistors, the bitlines can be precharged right after a row becomes activated. Therefore, only the sense amplifiers need to be precharged for a miss in most cases, taking an order of magnitude shorter time than the conventional deactivation process. Second, we show that this row-buffer decoupling enables internal DRAM μ-operations to be separated and recombined, which can be exploited by memory controllers to make the main memory system more energy efficient. Our experiments demonstrate that row-buffer decoupling improves the geometric mean of the instructions per cycle and MIPS^2/W by 14% and 29%, respectively, for memory-intensive SPEC CPU2006 applications.

1. Introduction

In pursuit of ideal memory (zero latency, unbounded bandwidth, and infinite capacity), DRAM architecture has evolved steadily under stringent cost constraints. Due to device scaling, multi-billion bits can be stored in a single DRAM device [38] and die stacking is on the horizon [31]. The number of pins per package has increased sluggishly, but advances in circuit technology have led to higher per-pin bandwidth achievable between the processor and memory packages, and emerging technologies [4, 13, 31] are expected to maintain this trend. However, main memory latency has not reduced much over technology generations because lengthy and narrow on-chip wires in deep sub-micron processes have high resistance and parasitic capacitance. On the other hand, identifying and transferring a few dozens of bits among billions of DRAM cells require huge fan-out and fan-in in control and data paths. It is important to reduce main memory latency to improve the performance of computer systems [37] as the memory footprint of popular applications [8, 11, 33] keeps increasing. This makes multi-level caches less effective in filtering memory requests, and the applications are often sensitive to memory access latency due to limited memory-level parallelism.

Memory controllers exploit the characteristics of DRAM architecture to enhance effective bandwidth and reduce average access latency [30, 32]. Modern DRAM devices have multiple banks, each of which consists of thousands of rows but has only one row buffer. To access data in a row, the row should be activated (opened) first. If it is already opened (a row-buffer hit), the activation process is not needed, reducing latency and energy. If the row buffer is occupied by another row (a row-buffer miss), the buffer should first be deactivated. Hasty deactivation needlessly re-opens rows for otherwise row-buffer hits while hindsight accompanies the deactivation process on the critical path of accessing data for row-buffer misses. Because the time to (de)activate a row is comparable to the time to read data from an opened row, around a dozen nanoseconds for modern DRAM devices [18, 34], it is crucial to properly manage the row buffers by correctly deciding when to close rows. A memory controller has a request queue and it decides when to close the row in a bank based on the pending requests to the bank. However, the memory controller often does not have sufficient pending requests to a bank because the number of banks per DRAM device keeps increasing, applications often have ample bank-level parallelism, and the size of the request queue is limited [2], all of which limit the number of requests per bank. Therefore, the memory controller has to rely on *prediction* in managing the row buffers.

Two popular row-buffer management policies are the open- and close-page policies [32], in which the former always keeps a row open and the latter closes it, both statically. Because memory access patterns vary significantly between applications and phases within each, there would be no clear winner between the two. More adaptive policies, which switch between open- and close-page policies based on a recent history

978-1-4799-4394-4/14/$31.00 © 2014 IEEE

of the row-buffer hit rates [17] and delay the precharge for a certain period [3, 21], have been proposed and exhibited a higher prediction accuracy than the static policies. However, their effectiveness is limited due to the dynamic nature of application behaviors, quantified in Section 5.

In this paper, we propose a novel DRAM microarchitecture that can obviate the need for any prediction instead of improving its accuracy. The open-page policy has a higher latency than the close-page policy on a row-buffer miss because the former needs to deactivate the currently active row. Even though deactivating the row buffer itself takes less than a nanosecond, precharging the bitlines (BLs) connected or *coupled* with the row buffer in conventional DRAM devices takes most of the deactivation time due to their dominance in total capacitance. By *decoupling* the row buffers from the BLs through adding isolation transistors between them, we can start BL precharging right after activating a row while making the row buffer hold the active row. This early-precharge policy, enabled by row-buffer decoupling, allows for BL precharging off the critical path in most row-buffer misses, achieving the latency of the open-page policy for row-buffer hits and a latency nearly identical to the close-page policy for misses which is the better of both static policies. The area overhead is minimal (0.8% for the modeled devices) because the isolation transistors can be an order of magnitude smaller than the bitline sense amplifiers (BLSAs), which function as row buffers, without affecting the activation time. The early-precharge policy incurs more frequent BL precharges for writes and hence consumes more energy, but the system-level static energy reduction due to improved performance outweighs this overhead.

We also show that this row-buffer decoupling enables richer design options by separating internal DRAM μ-operations and recombining them to DRAM commands, giving more flexibility in controlling the timings of the DRAM μ-operations. For example, a restore μ-operation, which is required as a part of an activate process due to the destructive nature of reading DRAM cells, can be delayed until a row-buffer miss by utilizing the isolation transistors, hoping for reducing the restore energy. We also augment the early-precharge policy by exposing activate and write DRAM μ-operations to the memory controller such that redundant restores and BL precharges are minimized in the cases of bursty writes to an open row. Our experiments on a simulated chip-multiprocessor system showed that the augmented early-precharge policy offered by row-buffer decoupling improves the geometric mean of the instructions per cycle (IPC) and MIPS^2/W by 24% and 43% over the open-page policy and by 14% and 29% over the adaptive row-buffer management policy [17], respectively, for nine memory-intensive SPEC CPU2006 applications.

In summary, this paper makes the following contributions:

- We identified that BLs dominate BLSAs, or row buffers, in capacitance. As a result, BL precharging takes most of the row deactivation time, which is the source of the performance penalty of the open-page policy.
- We propose a novel DRAM microarchitecture that minimizes the row deactivation time by decoupling the BLSAs from the BLs, allowing the BLSAs to hold an active row and the BLs to be precharged right after row activation, which is mostly off the critical path for row-buffer misses.
- We explore the design space of scheduling DRAM μ-operations enabled by row-buffer decoupling, and examine the design options of mitigating the excessive power consumption of the early-precharge scheme.
- We quantify the impact of row-buffer decoupling on the performance and energy efficiency of a simulated multicore system and demonstrate its superior performance over various system configurations.

2. Background and Motivation

2.1. State-Dependent DRAM Latency

The modern main memory system typically has one or more channels. A channel connects a memory controller with one or more dual in-line memory modules (DIMMs), each of which is divided into one or more ranks. A rank consists of multiple banks, where each bank can service a memory request independently. To provide the necessary bandwidth and capacity, a rank is often constructed with multiple DRAM devices connected in tandem; for a rank comprised of N DRAM devices, each DRAM device contributes to $1/N$ of the total bandwidth and capacity of a rank and a bank. Due to tight cost constraints, only one row (page) can be buffered (opened) in a row buffer per bank and actively service requests at a time.

When a memory controller receives a request, the latency for servicing the request significantly depends on the internal state of the bank servicing the request, apart from the queuing delay. First, consider a state in which the row buffer of the bank is invalidated after the datapath within the bank is precharged. For a given address, the corresponding row should be activated by an *activate* (`ACT`) command, which takes *tRCD*; the row is accessed and the data from the row are subsequently stored into the row buffer. Then, the data in the specified columns are accessed after *tAA*. The latency of such an access is $tRCD + tAA$. Second, consider a state in which the row stays open, maintaining the data from the previous request in the row buffer. If the current row address is the same as the previous one, a row activation (i.e., sending an `ACT` command) is not needed. Thus, the latency of such an access is simply *tAA*. However, if the current row address is not the same as the previous one, the previously opened row should be closed (deactivated) by a precharge (`PRE`) command, taking *tRP*, and then the row for the current address is activated. The latency of such a read is $tRP + tRCD + tAA$. Moreover, transitioning from one state to another takes a varied amount of time depending on the various DRAM timing and resource constraints. Thus, processor-memory interfaces, such as DDR3/4 [18, 34], LPDDR2/3 [28], and GDDR5 [19], *expose DRAM microarchitecture to a memory controller* so

that the memory controller can track internal bank states and make the best scheduling decision for the pending requests to minimize their average service latency.

2.2. Memory Access Scheduling and Page Management Challenges

A memory controller enqueues memory requests from processor cores and other sources in its request queue, tracks internal states of banks, and generates commands for the banks to service the requests, while considering various DRAM timing and resource constraints. The simplest memory controller (i.e., first-come-first-serve (FCFS)) checks the address of the oldest request in the queue and generates one or more commands to read or write data in the corresponding location depending on the internal state of the bank specified by a given address.

More advanced controllers check other recent requests in the queue and service a request by sending one or more commands if it satisfies DRAM resource and timing constraints. This effectively reorders the requests to improve channel utilization. Reordering may increase the access latency of some requests, but higher channel utilization mostly leads to lower latency for most requests on average. Typically, an older request, which has stayed longer in the queue, is considered with a higher priority than a younger request (first-ready FCFS (FR-FCFS) [32]). There can be other priority criteria, such as the number of requests from a certain CPU core [30], the access type (e.g., read over write), the access address (higher priority to the currently open rows), and the annotated criticality information. However, the degree of reordering can be limited to improve the quality of service, such as batching [30].

As discussed earlier, the latency and the number of commands to service a request heavily depend on the internal states of the banks. Hence, it is important for the memory controller to decide whether to close the currently open row that has just serviced a request. If any requests in the queue target the currently open row, it is better to keep the row open. In contrast, it is better to close the row by sending a `PRE` command to the bank if no request in the queue targets the currently open row. If a memory controller has no more enqueued request to the bank that has just serviced a request, it has to decide whether to close the currently open row after *predicting* the row address of future requests to the bank and act accordingly.

Misprediction penalty varies case by case, but its amount is significant because the precharge time (*tRP*) and the activate time (*tRCD*) are comparable to the address access time (*tAA*) taken from determination of the address to start of output of the read data. When it decides to keep a row open but the next request does not hit the same row in the bank, the subsequent request experiences an additional latency up to *tRP* because the datapath within the bank must be precharged before the next row is activated. When it decides to close a row but the next request hits the same row, that request experiences an additional latency of *tRCD* (or up to *tRC*, minimum `ACT` to `ACT` time to a bank, in the worst case when the next request arrives right after the decision is made and read-with-auto-precharge is used [34]). Moreover, such mis-prediction wastes DRAM energy due to one extra pair of `ACT` and `PRE`.

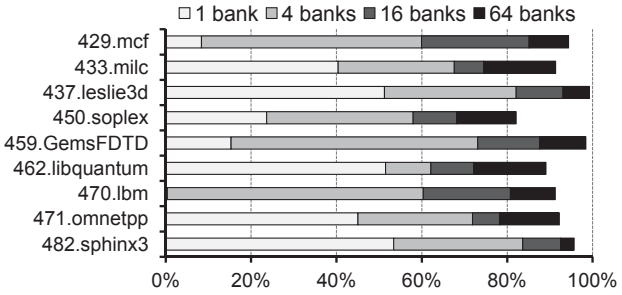

Figure 1: The rate that an access request needs predictions due to lack of information available for the controller on *spec-high* applications listed in Table 4.

Predicting whether to close a page is frequently inevitable in modern computer systems. If the request queue of a memory controller holds enough requests for at least two requests per bank, prediction is not needed at all. However, there are two reasons that this is unlikely to occur. First, applications often have ample bank-level parallelism (BLP). Thus, the number of enqueued requests to a specific bank can be small even though the applications have many pending requests in the queue. Second, the number of banks per DRAM device is increasing in contemporary DRAM architectures to narrow the performance gap between sequential and random memory accesses [17, 18, 31]. However, the request queue is an expensive resource and in the critical path [2]. Consequently, its size does not typically surpass few dozen slots. These two factors (i.e., increasing number of banks and limited request queue size) lead to even fewer enqueued requests per bank on average (i.e., higher probability of no or just one enqueued request to some banks). Therefore, unless an application has bursty memory requests with high spatial locality, which is not typically the case, prediction is needed more frequently.

Figure 1 shows the percentage of DRAM read and write requests that need predictions due to insufficient information (i.e., enqueued requests) available for the memory controller. The simulation methodology is detailed in Section 4. In many SPEC CPU2006 benchmark applications [11], predictions are needed for more than 80% of the requests even with 16 banks per memory controller. Furthermore, the percentage increases as the number of banks increases regardless of the applications.

The simplest prediction scheme is always to either close the row right after servicing a request or keep it open statically, which is similar to static branch prediction (i.e., always-taken or always-non-taken). The former and the latter are the close- and open-page row-buffer management policies, respectively. Because the spatial locality of main memory accesses varies significantly depending on the applications, there is no clear winner between the two shown in Figure 4 (more in Section 5).

Considering the dynamic nature of the application behaviors, however, it is challenging to achieve a high prediction

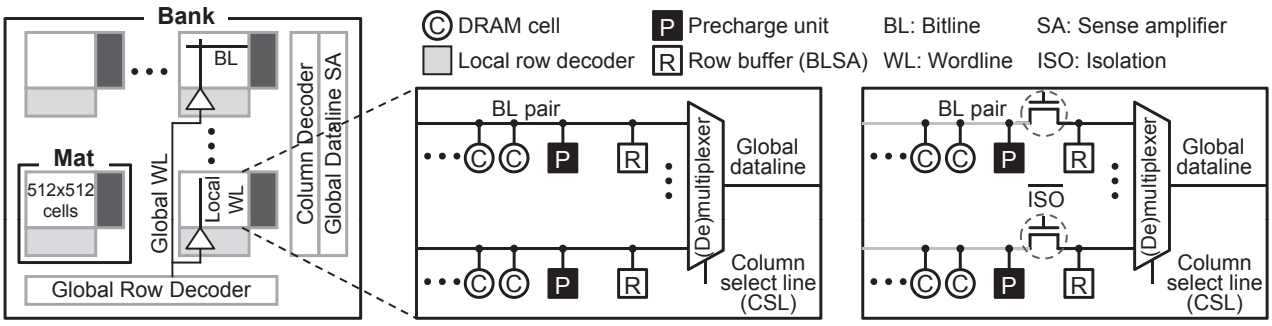

Figure 2: (a) A DRAM bank microarchitecture, (b) the datapath details of a mat for conventional DRAM devices, and (c) the datapath details of a mat with bitline sense amplifiers (row buffers) and (de)multiplexers decoupled from the bitlines and the DRAM cells by isolation transistors.

accuracy with either the close- or open-page policy. Consequently, various adaptive page-management policies, which ping-pong between the open- and close-page policies [15] and delay precharging [3, 21], have been proposed. Such adaptive policies can have a higher prediction accuracy than that of the static ones but with only limited success.

3. Row-Buffer Decoupling

Instead of lowering the frequency or improving the accuracy of prediction in managing row buffers, we propose a new DRAM architecture that *eliminates* the need for prediction. We identify that the performance difference between the open- and close-page policies originates from the fact that the row buffers lose data when a precharge process starts because the sense amplifiers that work as row buffers, DRAM cells that store data, and the bitlines (BLs) in between are *strongly coupled*. We propose modifying the internal DRAM structure to break the dependency between most of the precharge process and data retention by *decoupling the row buffers from the BLs and the cells*. In this section, we first review the pertinent details of the DRAM architecture, propose row-buffer decoupling, and explore more internal DRAM μ-operation scheduling policies enabled by decoupling, which enhances the performance and energy efficiency of main memory systems more than state-of-the-art row-buffer management policies.

3.1. Pertinent Details of DRAM Architecture to Explain Why Row Buffers are Coupled

A contemporary DRAM device employs a hierarchical structure to balance latency, bandwidth, energy consumption, and capacity under tight cost constraints. A DRAM device consists of multiple banks (e.g., 8 for DDR3), each of which is constituted with thousands of DRAM mats arranged in a two-dimensional fashion. As depicted in Figure 2(a), a DRAM mat, a unit of DRAM devices, consists of hundreds of rows and columns of cells (e.g., 512×512 bits), in which the rows and columns are connected to local wordlines (WLs) and BLs, respectively. A DRAM cell consists of an access transistor and a capacitor, which stores a bit of data by the voltage developed at the capacitor (i.e. Vdd means one and ground means zero, typically). Because the capacitance of a DRAM cell is much smaller than that of the connected BL, few BLs share a bitline sense amplifier (BLSA) to detect the value stored in the cells. Compared to the number of activated cells per mat, much fewer cells are read or written per request. Therefore, each mat has local datalines, which work as a datapath of the multiplexers/demultiplexers between BLSAs and global datalines. A global dataline is shared by all the mats in the same column of the bank.

With an `ACT` command from a memory controller, the global row decoder of a bank decodes the row address coincident with the `ACT` command and drives signals to the local row decoders of the corresponding mats through the global WLs. Subsequently, the local row decoders of the mats drive the local WLs coincident with the row of the bank to activate the row. Because very small cells ($6F^2$ for modern devices [38]) are employed to achieve a high integration density (i.e., low cost per bit), the developed BL voltage after the local WLs are driven is also very small, and cell states are destroyed as a consequence. Hence, a BLSA is used to detect, restore, and latch a value stored in an accessed cell, i.e., opening the corresponding DRAM row (page); a group of BLSAs that are involved in an `ACT` command is called a row buffer.

After an `ACT` command is sent, it takes *tRCD* for the column address coincident with a read (`RD`) or write (`WR`) command to be interpreted by the column decoder and delivered through the column select lines to control the multiplexers and demultiplexers. When the column address arrives at the specific mats through the column select lines, the BLSAs have not fully developed the BL voltage. Nonetheless, their values are large enough to be delivered through the local and global datalines for reads; after a `RD` command is sent, it takes *tAA* for the first chunk of the data (e.g., 8 bits for a ×8 device) to be available for transmission through a memory channel. BLSAs keep developing a voltage difference and restores the values back to the connected DRAM cells that have lost values; it takes *tRAS* after an `ACT` command to complete the activate and restore processes.

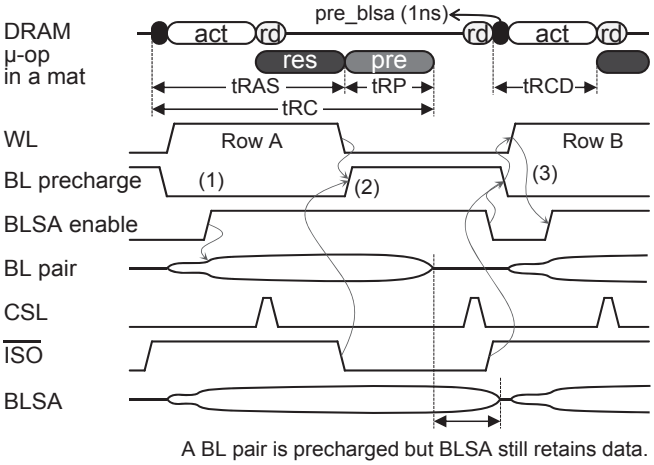

Figure 3: The timing diagram illustrating how the control and datapath signals within a mat supporting row-buffer decoupling are changed to process DRAM commands that activate a row, read it twice, and activate another row.

Precharge units precharge BLs and BLSAs to half the Vdd, the average of Vdd (one) or ground (zero) stored in a DRAM cell, to prepare for the next activate. This process is initiated by a PRE command or following a read or write process in the form of an auto precharge (for the close-page policy). In the current DRAM microarchitecture, the precharge units are all connected (or strongly coupled) with BLs, connectors to (de)multiplexers, and BLSAs as shown in Figure 2(b). Therefore, the following critical DRAM operations to read data from or write those to a row, which has not been activated yet, must be serialized: (i) the data of the currently active row are restored to the corresponding cells, (ii) all the BLs and BLSAs are precharged, and (iii) the new row is activated. Note that the precharge process takes considerable time (*tRP* is 11ns to 15ns for DDR3/4 [18, 34] devices) while destroying the data stored in the BLSAs (i.e., the row buffer). *Consequently, when to precharge significantly affects the latency of servicing subsequent memory requests in conventional DRAM devices.*

3.2. DRAM Architecture with Decoupled Row Buffers

We modify the DRAM mat microarchitecture to take the advantages of both the open-page and close-page management policies by breaking the dependency or coupling between data retention in the BLSAs and the BL precharge. We achieve this *decoupling* by adding isolation transistors to the BL pair connected to a BLSA as shown in Figure 2(c). When the isolation transistors are on, a DRAM mat can operate in the same way as a conventional one for reading and restoring the cell value. *When they are off, the data in the BLSAs can be read or updated by the global datalines through the (de)multiplexers while the BLs can be precharged in preparation for a following activate at the same time.*

The isolation transistors are controlled in such a way that the precharge time observed during a row-buffer miss is minimized as shown in Figure 3. Both the access transistor specified by a local WL driver and the isolation transistors are turned on such that the specified DRAM cell, the BL pair, and the BLSA are all connected during the activate and restore processes ((1) in Figure 3). After the data is restored to the cell, both the access transistor and the isolation transistors are turned off. This allows both the cell and the BLSA to retain the data of an open row while the BLs are precharged (2). Because the capacitance of a BL is an order of magnitude higher than that of a BLSA [1], the time to precharge the BLs dominates the *tRP* of a conventional DRAM mat. Therefore, only a small portion of the *tRP* of a conventional DRAM mat is needed to precharge the BLSA when a new row is activated by disabling the BLSA and turning on the isolation transistor. Then, the precharge unit is disabled and the access transistor corresponding to the new row is turned on initiating a new activate process, which is followed by enabling the BLSA after charge sharing between the BL and the new cell develops a voltage difference which is enough for sensing (3).

This *early-precharge* policy enabled by row-buffer decoupling does not save the read latency if the DRAM cycle time (*tRC*) becomes the limiting factor because the activate, restore, and precharge processes are still needed per active row switch. However, a memory controller often has multiple ranks, each of which consists of several banks in modern main memory systems, and there are often multiple accesses per open DRAM row. Therefore, the average interval between two activates on a bank would be much longer than the *tRC* on many applications, where row-buffer decoupling does reduce access latency for row-buffer misses. Because the time to precharge the BLSAs is so small, we do not dedicate a DRAM command for it and instead, we include it as part of an activate process.

Decoupling the row buffers using the isolation transistors is feasible and their area overhead is minimal. We leverage the concept of using isolation transistors, which reduce the delay and power consumption, from SRAM designs [39]. In SRAM devices, the isolation transistors disconnect a BLSA from a pair of complementary BLs right after a specified SRAM cell develops just enough voltage for the BLSA to sense a stored value. This in turn prevents the pair of BL from swinging rail-to-rail, reducing unnecessary power consumption for the next precharge cycle. Furthermore, because the BLSA is disconnected from the BL exhibiting a large capacitance, it can swing to a desired voltage level faster than the BLSA that is still connected to the BLs even after a sufficient voltage difference is developed. *Although isolation transistors have been used for DRAM devices before, their purposes was simply to make multiple BLs share a BLSA to lower the area overhead of bulky BLSAs [9] and they have not been exploited for page management like in our proposal.* Tiered-latency DRAM [25] also used isolation transistors, but those were used to provide asymmetry in accessing DRAM rows, unlike our row-buffer decoupling. We set the size of an isolation transistor pair as

[1] Bitline capacitance is 12× higher than BLSA capacitance in the 28nm DRAM process we modeled for simulation, which is specified in Section 3.4.

one tenth of the size of a BLSA, as suggested by Gealow [9] and in tiered-latency DRAM [25], which is large enough to be turned on and off within few tenths of a nanosecond. Therefore, the area overhead from adding the isolation transistors and their supporting circuitry is low at 0.8% for the DRAM process we specify in Section 3.4.

The energy overhead of the isolation transistors is also insignificant. An isolation transistor is turned on and off once per active row switch, like the access transistor of a cell, a precharge unit, and a BLSA. The driver for the isolation transistors is much smaller than the driver for the BLSAs. Because the DRAM device gets larger, the lengths of the global and inter-bank datalines increase, and their driver sizes are adjusted to maintain the data transfer latency, which is properly considered during the energy modeling in Section 3.4.

So far, we have focused on DRAM reads, where data restored in DRAM cells need not be updated. In contrast, a DRAM write updates the data latched in the BLSAs using write drivers. Because the DRAM cells are disconnected from the BLSAs during the write in the case of the early-precharge policy, the cells must be reconnected to the BLSAs, updated (restored), and disconnected again, after which the BLs can be precharged to reduce the active row switch time. Even though the energy to update the cells during a write is lower than the energy consumed during a restore process, the update and precharge energy is not negligible and clearly an overhead. One way to alleviate this overhead is to keep the cells and the BLSAs connected once a `WR` command is applied to an open row. Then, the active row switch time would be the same as the case of the open-page policy, but because writes are less frequent than reads for many applications [7], their impact on average memory access latency could be limited. Another way to alleviate the energy overhead is to make the memory controller detect consecutive writes on an open row. Then, all but the last write can skip the update and precharge processes. We elaborate on these extensions in the following section.

3.3. Scheduling Internal DRAM μ-operations

Row-buffer decoupling not only enables early precharges but also provides more freedom in controlling the timings of internal DRAM μ-operations. A DRAM command prompts one or more μ-operations within the specified DRAM devices, which is similar to μ-ops in CPUs. For conventional DRAM devices, an `ACT` command corresponds to latching data of the specified row at the BLSAs (a μ-operation called `act`) followed by restoring the data back to the original cells (`res`). A `WR` command overwrites a portion of data at the BLSAs (`wr`) then updates the overwritten values to the corresponding DRAM cells (`res`). A `PRE` command precharges both BLs (`pre_bl`) and BLSAs (`pre_blsa`). A `RD` command is not further divided and corresponds to a `rd` μ-operation. We use capital letters for the explicit DRAM commands and lower-case letters for the μ-operations. With the isolation transistors, `pre_bl` can be a part of `ACT` and `WR` to save a row-buffer miss latency discussed in Section 3.2 for the early-precharge policy. In contrast, if we enforce an `ACT` not to do `res` and delay it until `act` fully develops BLSAs, which is enabled by the isolation transistors, we can reduce the `res` energy. This is because BLSAs are more power efficient when used as drivers instead of performing sensing and driving at the same time. In addition, if an entire row gets overwritten after being activated or the row is read but not accessed any more, `res` can be omitted.

CMD	μ-op	Dres	Eres (Ref)	[Epre, Dres]	[Epre, Eres]	[Epre, Epre]
`ACT`	`pre_blsa`			√	√	√
	`act`	√	√	√	√	√
	`res`		√	√	√	√
	`pre_bl`			√	√	√
`WR`	`wr`	√	√	√	√	√
	`res`		√		√	√
	`pre_bl`					√
`PRE` (no `WR` after `ACT`)	`res`	√				
	`pre_bl`	√	√	N/A	N/A	N/A
	`pre_blsa`	√	√			
`PRE` (1+ `WR` after `ACT`)	`res`	√		√	√	
	`pre_bl`	√	√	√	√	N/A
	`pre_blsa`	√	√			

Table 1: Mapping between DRAM commands and the matching μ-operations for several scheduling policies enabled by row-buffer decoupling. √ indicates that the μ-op is the part of the DRAM command in the same row for a certain policy.

The design space of scheduling the DRAM μ-operations is huge, and the performance and energy efficiency of the system can vary significantly depending on the interaction between scheduling policies and memory access patterns. However, the early-precharge policy performs best among the possible scheduling options because delaying `res` or `pre` might save energy but does not reduce the latency for row-buffer misses. Therefore, if an application is less sensitive to main memory latency, a delayed-restore policy could improve the system-level energy efficiency with negligible degradation in performance. Even though it could be interesting to devise adaptive schemes for scheduling the μ-operations, such as adopting reinforcement learning [16, 29], it is beyond the scope of this paper and will be considered in possible future work. Table 1 enumerates the mapping between DRAM commands and the corresponding μ-operations for several representative policies. For both `ACT` and `WR`, we have three options: 1) neither `res` nor `pre_bl` (Dres), 2) only `res` after `act` or `wr` (Eres), and 3) `res` then `pre_bl` (Epre). We use the notation [policy for `ACT`, policy for `WR`] or just policy in case `ACT` and `WR` have the same policy. If either `ACT` or `WR` requires `pre_bl`, `ACT` should start with `pre_blsa` to disable the BLSAs before the normal activation process begins. The `PRE` command can be omitted for [Epre, *] if there is no `WR` after the row gets activated. Epre does not need `PRE` at all. Eres operates the same as the open-page policy

Parameter	Reference	Delayed restore	Early precharge
tRCD	14ns	14ns	15ns
tAA	14ns	14ns	14ns
tRC	49ns	54ns	50ns
tWR	15ns	10ns	29ns
tRP	14ns	19ns	N/A
E_{ACT}	42.5nJ	14.1nJ	50.1nJ
E_{RD}	7.0nJ	7.1nJ	7.1nJ
E_{WR}	7.0nJ	7.0nJ	32.2nJ
E_{PRE}	7.5nJ	12.5nJ	N/A

Table 2: Latency and energy parameters of the primary DRAM operations for the rank without decoupling (reference) and the rank with decoupled row buffers employing the delayed-restore (Dres) and early-precharge (Epre) policies.

of conventional DRAM devices. On top of these policies, we also examine the option of populating two kinds of `ACT` and `WR`, one with `res` and the other without it, and letting the memory controller use them properly to precharge the BLs only once when there are multiple writes to an open bank in the request queue for Epre. We call this new policy an augmented early-precharge scheme or a-Epre. In Section 5, we compare the performance of Dres, Epre, and a-Epre.

3.4. Quantifying the Row-Buffer Decoupling Overhead through SPICE Modeling

To quantify the overheads and savings of row-buffer decoupling, we modeled the power and timing of internal DRAM components and operations using SPICE simulation, and we obtained the inter-die I/O power values of the processor-memory interfaces from the latest DDR3 specifications [34]. We assumed the following: 28nm DRAM process, 3 metal layers, 8Gb and 8 banks per device, 8KB per row, 16GB/s per channel, and $6F^2$ DRAM cells for the SPICE modeling with a modified version of the PTM low-power model [41]. Row-buffer decoupling slightly increases the DRAM die area (by 0.8%), which in turn increases the on-chip datapath length. We increased the driver strength to keep the read and write latency same as the reference device and applied the corresponding energy increase during DRAM reads and writes.

The acquired latency and the energy consumption values of major DRAM operations for the reference (without decoupling), Dres, and Epre ranks are summarized in Table 2. The read time (*tAA*) is the same for all the DRAM types. Dres has a higher cycle time (*tRC*) than that of the others because `res` is the part of a `PRE` command. *tRCD* is 15ns for Epre and 14ns for others due to the overhead from disabling and precharging the BLSAs. The write recovery time (*tWR*) is different because Dres dispenses with `res` while Epre entails both `res` and `pre` for a write. The precharge time (*tRP*), is 14ns for the reference rank and 19ns for Dres because `res` is needed. Similar trends are observed for energy consumption. Epre consumes more energy for `WR` than other types because it does both `res` and `pre`. The sum of the `ACT` and `PRE` energy is the lowest for Dres because BLSAs have lower load capacitance during sensing and work as drivers during restore as explained in Section 3.2.

Resource	Value
Number of cores	16
Number of memory controllers	4
Coherence policy	MESI
Coherence implementation	Reverse directory
Per core:	
Frequency	3GHz
Issue policy	Out-of-Order
Issue/commit width	4 / 4
L1 I/D cache size/associativity	16KB / 4
L2 cache size/associativity	1MB / 16
L1, L2 cache line size	64B
Per memory controller:	
Number of channels per controller	1
Bandwidth per channel	16GB/s
Number of ranks per channel	2
Capacity per rank	8GB
ECC	SECDED
Scheduling policy	PAR-BS [30]
Request queue size	32
DRAM refresh interval (*tREFI*)	7.8μs

Table 3: Default parameters of the simulated multicore system.

4. Experimental Setup

We simulated a chip-multiprocessor system with multiple memory controllers to evaluate the system-level impact of row-buffer decoupling with the default parameters summarized in Table 3. Memory addresses were interleaved in the following way: DRAM row, bank, rank, memory channel, and column from the most significant address bit to the least significant bit (so called page interleaving) by default. Other interleaving schemes are compared in Section 5.3. We modified McSimA+ [1] to support row-buffer decoupling for performance simulation. McPAT [26] was used for modeling the power and timing values of the cores, caches, and memory controllers assuming a 22nm logic process.

The SPEC CPU2006 [11], SPLASH-2 [40], and PARSEC [5] benchmark suites were used for evaluation. We identified the representative phases for SPEC CPU2006 applications using Simpoint [35]. Per application, we chose the top-4 slices in weight, each of which includes 100 million instructions. As for SPLASH-2 and PARSEC, we simulated regions of interest and used the datasets listed in [26]. We classified the SPEC CPU2006 applications into three groups (spec-low, spec-med, and spec-high) based on the main memory accesses per kilo-instructions (MAPKI) [11]. Table 4 shows four mixtures we created as multiprogrammed workloads. mix-high is composed of spec-high applications; mix-close consists of the applications with relatively high row-buffer conflict rates favoring the close-page policy; mix-open consists of the ones

Mixes	400.perlbench	401.bzip2	416.gamess	435.gromacs	444.namd	445.gobmk	447.dealII	453.povray	454.calculix	456.hmmer	403.gcc	410.bwaves	434.zeusmp	436.cactusADM	458.sjeng	464.h264ref	465.tonto	473.astar	481.wrf	483.xalancbmk	429.mcf	433.milc	437.leslie3d	450.soplex	459.GemsFDTD	462.libquantum	470.lbm	471.omnetpp	482.sphinx3
	spec-low										spec-med									spec-high									
mix-high																				2	2	2	2	2	2	2	1	1
mix-open				1	1			1	1		1				1		1		1		2	1	2		1			2
mix-close			2							1			2	2	1			2		2				2			2		
mix-blend	1	1					1	1				1	1		1	1		1		1	1			1		1	1	1	

Table 4: We categorize the SPEC CPU2006 applications into 3 groups based on the main memory accesses per kilo instructions (MAPKIs). The numbers in the table mean the number of cores that run the specific applications for a certain mixture of multiprogrammed workloads.

with low row-buffer conflict rates, and mix-blend is composed of four applications from spec-low, six from spec-med, and six from spec-high. For each mixture, each core runs a simulation point, and the number of simulation points used per application is specified in Table 4. A weighted speedup approach [36] was used to evaluate the performance of the multiprogrammed workloads. To compute the weighted speedup, we first determined the relative performance of each application and the ratio of its IPC in a configuration for a multiprogrammed workload to the IPC in a standalone execution configuration, and accumulated the relative performance of all the applications belonging to the mixture.

5. Evaluation

We evaluated the system-level impacts of row-buffer decoupling on the simulated multicore system. We first used single-threaded workloads to quantify the benefits of employing decoupled row buffers with an average main memory read latency, IPC (instructions per cycle), and MIPS^2/W (square of million instructions per second per Watt)[2]. Then, we populated multiple memory channels and cores to quantify the effectiveness of row-buffer decoupling on multi-programmed and multi-threaded workloads and to demonstrate its robustness over various system configurations.

5.1. The Impact of Row-Buffer Decoupling on Single-Threaded Workloads

Figure 4 shows the average memory read latency, relative IPC, and relative MIPS^2/W values of the SPEC CPU2006 applications for systems with and without decoupled row buffers. The average values for all the SPEC CPU2006 benchmark applications (spec-all) and their memory-bandwidth-demanding subset (spec-high specified in Table 4) are also presented. Six main memory system policies were used for the evaluation. Among the ones without row-buffer decoupling, open, close, and adapt employ the open-page, close-page, and adaptive row-buffer management policies, respectively. As for the system with decoupled row buffers, the delayed-restore (Dres) policy was evaluated as well as the early-precharge (Epre) policy and its augmentation that utilizes ACT and WR without restore in the case of consecutive writes (a-Epre). For close, we make a row stay open until the *tRAS* timing constraint is met, as suggested by the minimalist-open policy [21], instead of closing it with an auto-precharge command. At the end of an epoch, adapt ping-pongs between the open- and close-page policies based on the row-buffer hit rate of the current epoch [17]. We set 100 DRAM accesses as the size of an epoch. MIPS^2/W is inversely proportional to the energy-delay product. We populated one core and one memory controller for the SPEC CPU2006 single-threaded workloads to stress the main memory bandwidth. open is the baseline policy for each application. The number within the round brackets in the label per application is the absolute IPC on open.

We made the following observations from the experiments on the single-threaded workloads. **(1) The early-precharge policy (Epre) enabled by row-buffer decoupling substantially improves both the performance and energy efficiency of the simulated system compared to the policies without decoupled row buffers.** Among the main memory bandwidth demanding SPEC CPU2006 applications, 429.mcf, 459.GemsFDTD, and 471.omnetpp had a lower memory read latency and higher IPC for close rather than open because they have relatively higher row-buffer conflict rates on the simulated system. The other six applications grouped under spec-high favor open. adapt successfully traces the transition of the row-buffer conflict rates depending on applications and their phases so that it gives a higher IPC and MIPS^2/W than open and close even though the values are often similar to the better of the two. More importantly, Epre consistently outperforms adapt in the average read time, IPC, and MIPS^2/W, which shows the effectiveness of overlapping the bitline precharge process and row-buffer retention through decoupling. On average, for main memory bandwidth demanding applications (spec-high), Epre gives 24%, 27%, and 14% higher IPC values than that of open, close, and adapt, respectively. Compared to adapt, Epre reduces the average main memory read latency by 20% and improves the MIPS^2/W by 16%.

[2]It is often a metric of choice to compare power efficiency of high-end computing systems along with MIPS^3/W [6].

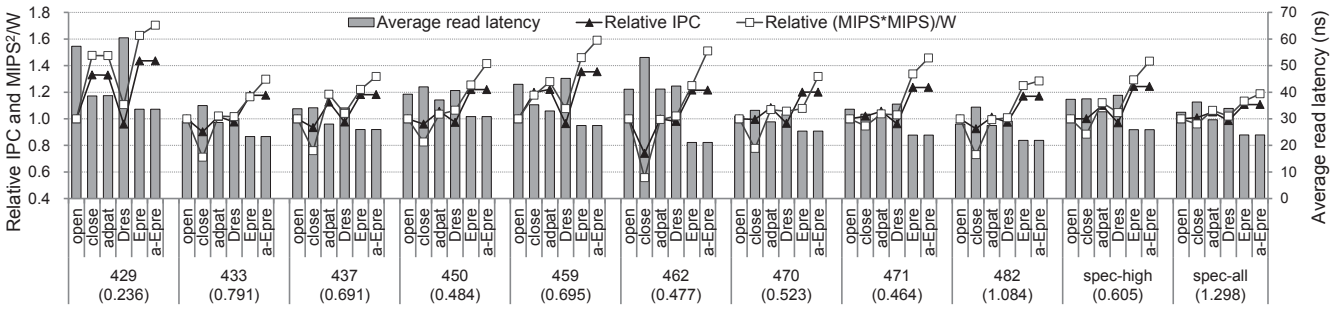

Figure 4: The average memory read latency, relative IPC, and MIPS^2/W values of SPEC CPU2006 applications and their averages for the open-page, close-page, and adaptive management policies without decoupled row buffers and the delayed-restore (Dres), early-precharge (Epre), and augmented-early-precharge (a-Epre) policies with row-buffer decoupling. The number within the round brackets in the label per application is the absolute IPC on open.

(2) The energy efficiency of the early-precharge policy can further be improved by allowing the memory controller to exploit the ACT and PRE commands without restore. Even though Epre is consistently better than adapt in IPC and MIPS^2/W, the gap in MIPS^2/W between the two is small on 437.leslie3d and 470.lbm. This is because Epre consumes restore and precharge energy for every write. The augmented early-precharge policy, a-Epre, saves the restore and precharge energy by letting the memory controller detect the cases when the first access after row activation is write and there are multiple writes piled up in the request queue to an open row as described in Section 3.3. Thus, a-Epre performs equally to Epre but also improves MIPS^2/W by as high as 23% on 470.lbm, and on average, 12% of the applications categorized as spec-high, respectively, compared to Epre. Note that the delayed-restore policy, Dres, is not as effective as Epre or adapt because it has higher latency penalties for row-buffer misses even though it conserves DRAM dynamic energy. Because the additional static energy dissipation from the CPUs and the DRAM devices due to a longer execution time outweigh the energy savings from delaying restore operations, Dres does not match Epre in MIPS^2/W either.

(3) The impact of row-buffer decoupling on system-level performance and energy efficiency is diminished for less memory bandwidth demanding applications, but they also experience a noticeable reduction in the average read latency. Compared to adapt, Epre improves IPC by 14% on average for spec-high applications, but only 6% on average for all the SPEC CPU2006 applications (spec-all) because a reduction in memory accesses is less critical to the performance of applications with lower MAPKI values. However, the average main memory read latency of Epre is 20% and 19% lower than that of adapt for spec-high and spec-all, respectively, which is not very different. All these analyses show that neither the open- nor close-page policy clearly outperforms the other, and the adaptive policy is better than both, but it is only as good as the early-precharge policies in reducing the memory access time while its impact on system-level performance and energy efficiency depends on the main memory bandwidth demands of applications.

5.2. The Impact of Row-Buffer Decoupling on Multi-threaded and Multi-Programmed Workloads

Row-buffer decoupling is also effective on multi-programmed and multi-threaded workloads. Figure 5 shows the average memory read latency, relative IPC, and relative MIPS^2/W values of multi-programmed and multi-threaded workloads for the main memory system employing the open, close, and adapt policies without row-buffer decoupling and for the system employing Dres, Epre, and a-Epre policies with decoupling. Compared to the system simulated for the single-threaded workloads, the system for the multi-programmed and multi-threaded workloads has 16 times more cores and 4 times more memory controllers. Therefore, the number of banks per application decreases for the multi-programmed workloads, which increases the row-buffer miss rates. Among the multi-programmed workloads specified in Table 4, mix-high, mix-close, and mix-blend favor close while mix-open favors open. adapt performs better than the worse of the open and close, but it is less effective compared to the case of single-threaded workloads because main memory requests from different cores, which often have different access patterns, interfere with each other and make the adaptive decision made by the controllers less accurate. Because row-buffer decoupling does not rely on prediction, a-Epre is consistently superior to the policies for the system without the decoupled row buffers, which is better than adapt by 12% and 19% in IPC and MIPS^2/W, respectively, for mix-blend.

Even though the sensitivity for memory access latency varies, the early-precharge policies improve the IPC and MIPS^2/W of SPLASH-2 and PARSEC applications. For example, on LU, a SPLASH-2 application, a-Epre is 14% and 23% better than adapt in IPC and MIPS^2/W, respectively. The STREAM benchmark [27] is an interesting application because its performance is much more sensitive to main memory bandwidth than the latency, and it has high spatial locality in memory accesses. Therefore, all the tested policies perform similarly and close, and Apre has the worse energy efficiency than that of the others due to frequent restore and precharge μ-operations for WRs. Dres is more energy efficient than the

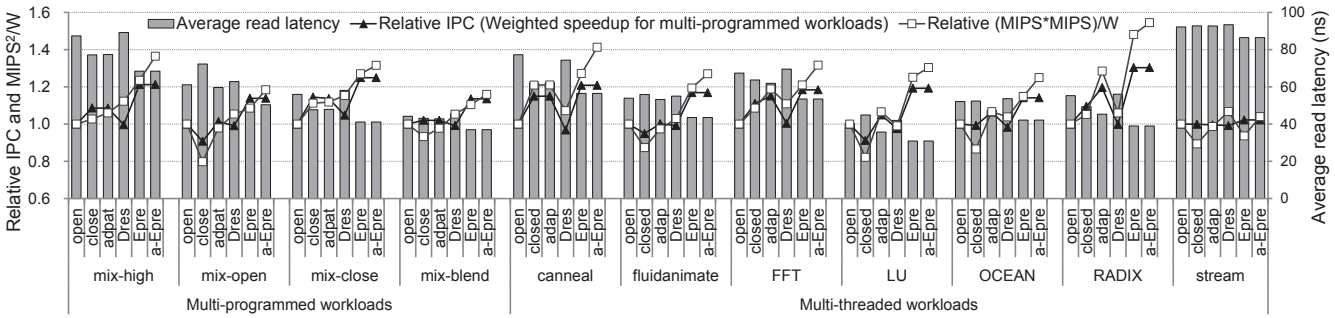

Figure 5: The average memory read latency, relative IPC, and MIPS^2/W values of multi-programmed (specified in Table 4) and multi-threaded workloads for the open-page, close-page, and adaptive policies without row-buffer decoupling and the delayed-restore (Dres), early-precharge (Epre), and augmented-early-precharge (a-Epre) policies with decoupling.

others because it consumes less restore energy than the other policies as explained in Section 3.3. The energy efficiency of a-Epre is close to that of Dres and better than Epre because the memory controller of a-Epre successfully detects the stream of memory writes and minimizes the restore and precharge μ-operations, correspondingly.

5.3. Sensitivity Studies

The superiority of the DRAM devices with decoupled row buffers remains unchanged even if we apply data prefetching and vary the address interleaving scheme, number of memory channels, and number of ranks per channel. Figure 6 shows the relative IPC and MIPS^2/W of the open, close, adapt, and a-Epre over various memory-system configurations for mix-high (Figure 6(a)) and mix-blend (Figure 6(b)). We use the notation (interleave, N_{MC}, N_{rank}, T/F), for which interleave is either page interleaving (P), cache line interleaving (L), or interleaving at the granularity of 4 cache lines (4L); the number of memory channels is either 4, 8, or 16 (N_{MC} = 4, 8, 16); the number of ranks per channel is either 2, 4, or 8 (N_{rank} = 2, 4, 8), and T/F means whether a hardware prefetcher [20] is used (T) or not (F). Bold fonts are used to highlight the parameters deviated from the default values. The open-page policy with (P, 4, 2, F) is the baseline per workload.

The cache-line-interleaving scheme relatively favors the close-page policy while the page-interleaving scheme favors the open-page policy in IPC and MIPS^2/W. The 4-cache-line-interleaving scheme, which is suggested by [21, 31], performs in-between. a-Epre outperforms the policies without decoupling regardless of the interleaving schemes. With more memory controllers ((P, *, 2, F)), both the IPC and MIPS^2/W values are improved for all the page-management policies, but the policy that provides the highest performance and energy efficiency is always a-Epre. When we increase the number of ranks per channel from 2 to 4 and 8, the IPC values first increase then decrease because more banks per channel lowers row-buffer conflicts, but the overhead of switching the ownership of memory channels (*tRTRS* [17]) negates the impact of reduced row-buffer conflicts when N_{rank} is large. Because populating more ranks implies that more DRAM devices dissipate static power and more I/O energy is needed per bit of data transfer due to inferior signal integrity, system power consumption increases resulting in a worse MIPS^2/W. Prefetching increases both the IPC and DRAM power consumption because it decreases the average memory access time with decent prefetch hit rates, but at the same time wastes main memory bandwidth on prefetch misses. For the tested workloads, the advantages of prefetching outweigh the disadvantages and hence, both IPC and MIPS^2/W are improved. More importantly, prefetching is beneficial to the page-management policies with row-buffer decoupling as well so that a-Epre is still the one with the highest performance and energy efficiency.

6. Related Work

Microarchitectures pursuing higher DRAM throughput: The DRAM industry had been highly sensitive to cost and reluctant to adopting new architectures. However, as DRAM device scaling becomes harder, more drastic changes to the DRAM microarchitecture have been suggested both by the research community and industry. Exploiting subarray-level parallelism [24], a DRAM bank architecture supports parallel accesses to different subarrays constituting a bank. This allows the memory controller to overlap accesses to the same bank, effectively reducing access latency. Gulur et al. [10] observed that memory access interferences by concurrently running applications degrade row-buffer locality and proposed to divide an existing row buffer into multiple sub-row buffers. This DRAM architecture not only increases the hit rate of the row buffers (i.e., lower latency) but also decreases its energy consumption.

CHARM [37] reorganizes a subset of DRAM banks to have more BLSAs and allows banks to be accessed non-uniformly to reduce the average access latency. The tiered-latency DRAM architecture achieves both low latency and low cost per bit by splitting each long bitline into two shorter segments using an isolation device [25]. This allows one segment closer to the BLSA to be accessed with the latency of a short BL after the other segment is disconnected (isolated) from the BL and its BLSA, whereas we used isolation transistors to better manage row buffers. The hybrid memory cube is an-

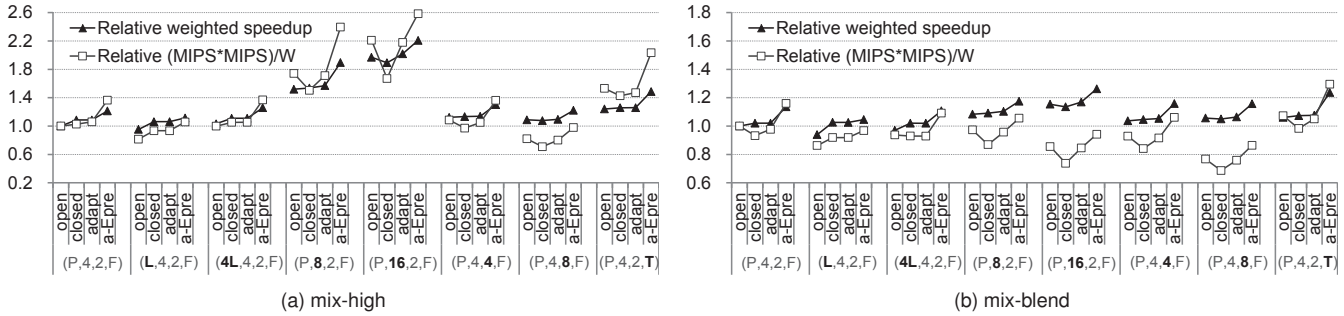

Figure 6: The relative IPC and MIPS^2/W of open, close, adapt, and a-Epre on mix-high (a) and mix-blend (b). We use the notation (interleave, N_{MC}, N_{rank}, T/F), where interleave is page (P), line (L), and 4-line (4L) interleaving, 4, 8, and 16 memory channels (N_{MC}), 2, 4, and 8 ranks per channel (N_{rank}), and T/F means if hardware prefetching is used (T) or not (F).

other new DRAM architecture that integrates multiple DRAM dies with a memory controller through silicon vias (i.e., 3D stacking technology) [31]. This DRAM architecture can reduce energy consumption and improve bandwidth compared to conventional DRAM devices, but it has a latency overhead because it adopts a packetized interface, and its high-speed links have (de)serialization overheads. Row-buffer decoupling is complementary to the above architectures, and can be used on top of them to further reduce main memory latency.

The cache DRAM architecture [12] added bulky SRAMs for caching frequently accessed data. Hwang et al. [14] exploited these SRAMs to retain an active page and let an `ACT` command skip a `res` µ-ops in embedded DRAMs. By contrast, we exploited existing BLSAs to minimize the area overhead.

Memory access schedulers: Many studies have focused on memory scheduling to improve either bandwidth utilization (i.e., higher DRAM throughput), fairness, or both. To achieve higher bandwidth utilization, the FR-FCFS scheduling policy prioritizes row-buffer hits [32]. FR-FCFS has been readily accepted for single-core processors, but it does not guarantee any fairness when used in a chip-multiprocessor (CMP) environment because it tends to favor applications with high row locality. Recently proposed policies attempt to make a memory controller aware of the characteristics of diverse applications being executed, and provide fairness to all applications. The parallelism-aware batch scheduling (PAR-BS) scheme issues requests in batches to provide fairness and tries to maintain the original bank-level parallelism of an application [30]. The adaptive per-thread least-attained-service scheduler (ATLAS) prioritizes requests from applications that have received the least service from memory controllers [22]. Thread cluster memory scheduling (TCM) improves both bandwidth utilization and fairness by dividing applications into two groups, i.e. latency-sensitive and bandwidth-sensitive cluster, based on bank-level parallelism, row locality, and the memory intensity of the applications being executed [23]. As an adaptive memory scheduling approach, multi-objective reconfigurable self-optimizing memory scheduler (MORSE) adjusts the parameters of a self-optimizing memory scheduler [16] for various objectives such as power or energy instead of just bandwidth utilization through various techniques [29]. In this paper, we leveraged the PAR-BS scheme, but all these scheduling policies are orthogonal to row-buffer decoupling because the schedulers make decisions based on the pending requests in the queue while our row-buffer decoupling targets the case in which the information provided by the queue is insufficient.

DRAM page-management policies: Minimalist open-page policy [21] interleaves memory accesses at a sub-row granularity across channels and banks to minimize memory access interference between applications contending for access to the main memory system with an open-page policy, increasing bandwidth utilization and improving the fairness for individual applications in a CMP environment. An adaptive open- and close-page policy, which was implemented in commercial Intel processors, attempts to improve both bandwidth utilization and access latency by adaptively choosing a page policy depending on the memory access characteristics of applications being concurrently executed [15]. We have already compared the adaptive open- and close-page policy with row-buffer decoupling and studied the impact of the address interleaving schemes.

7. Conclusion

We have proposed a new DRAM microarchitecture for the main memory that decouples a bitline sense amplifier (BLSA), which works as a row buffer, from the corresponding bitline pair using isolation transistors. The proposal is based on the key observations that 1) the number of banks per memory channel keeps increasing so that the DRAM cycle time (minimal interval between row activates at a bank) has little influence on access latency; 2) the capacitance of a bitline is significantly larger than that of a BLSA so that bitline precharging takes most of the precharge time on modern DRAM devices, and 3) the precharge process is on the critical path in the case of a row-buffer miss.

The decoupling allows BLSAs to retain an open row and the bitlines to be precharged right after the row becomes activated so that the bitline precharging is off the critical path of the following row-buffer miss in most cases. On a row-buffer

conflict, only the BLSAs need to be precharged, taking significantly less time than that of the conventional precharge process. Therefore, the early-precharge policy with decoupled row buffers perform as good as the open-row policy on a row-buffer hit, almost the same as the close-page policy on a miss, and better than both policies when row-buffer hits and misses are mixed. Row-buffer decoupling incurs less than 1% area overhead, but the early-precharge policy causes excessive bitline precharge operations for writes. We mitigate this energy overhead by making the bitline precharge optional for the activate and write commands, enabled by row-buffer decoupling that separate and recombine internal DRAM μ-operations, and letting the memory controller choose either type based on the requests stacked in its queue. The simulation results show that row-buffer decoupling improves the instruction per cycle and MIPS^2/W by 24% and 43%, respectively, compared to the open-page policy on average for nine main memory bandwidth demanding SPEC CPU2006 applications.

Acknowledgements

This material is based on work supported by Samsung Electronics (SLSI-201301GE001 and 490-20130029), by the Basic Science Research Program through the NRF funded by the MSIP (2012R1A1B4003447), by IDEC (EDA Tool), by generous grants from NSF (CCF-1016262). Nam Sung Kim has a financial interest in AMD.

References

[1] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *ISPASS*, Apr 2013.

[2] R. Ausavarungnirun, K. Chang, L. Subramanian, G. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, Jun 2012.

[3] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," in *ISCA*, Jun 2000.

[4] S. Beamer, C. Sun, Y.-J. Kwon, A. Joshi, C. Batten, V. Stojanovic, and K. Asanovic, "Re-Architecting DRAM Memory Systems with Monolithically Integrated Silicon Photonics," in *ISCA*, Jun 2010.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, Oct 2008.

[6] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *Micro, IEEE*, vol. 20, no. 6, 2000.

[7] H. Choi, J. Lee, and W. Sung, "Memory Access Pattern-Aware DRAM Performance Model for Multi-Core Systems," in *ISPASS*, Apr 2011.

[8] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA Database: Data Management for Modern Business Applications," *SIGMOD Record*, vol. 40, no. 4, 2012.

[9] J. C. Gealow, "Impact of Processing Technology on DRAM Sense Amplifier Design," Ph.D. dissertation, Massachusetts Institute of Technology, 1990.

[10] N. D. Gulur, R. Manikantan, M. Mehendale, and R. Govindarajan, "Multiple Sub-Row Buffers in DRAM: Unlocking Performance and Energy Improvement Opportunities," in *ICS*, Jun 2012.

[11] J. L. Henning, "SPEC CPU2006 Memory Footprint," *Computer Architecture News*, vol. 35, no. 1, 2007.

[12] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima, "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," *Micro, IEEE*, vol. 10, no. 2, 1990.

[13] R. Ho, P. Amberg, E. Chang, P. Koka, J. Lexau, G. Li, F. Y. Liu, H. Schwetman, I. Shubin, H. D. Thacker, X. Zheng, J. E. Cunningham, and A. V. Krishnamoorthy, "Silicon Photonic Interconnects for Large-Scale Computer Systems," *Micro, IEEE*, vol. 33, no. 1, 2013.

[14] C.-L. Hwang, T. Kirihata, M. Wordeman, J. Fifield, D. Storaska, D. Pontius, G. Fredeman, B. Ji, S. Tomashot, and S. Dhong, "A 2.9ns Random Access Cycle Embedded DRAM with a Destructive-Read Architecture," in *Symposium on VLSI Circuits*, Jun 2002.

[15] Intel, *Intel® Xeon® Processor 7500 Series Datasheet*, Mar 2010.

[16] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, Jun 2008.

[17] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc, 2007.

[18] JEDEC, *DDR4 SDRAM Specification*, Sep 2012.

[19] JEDEC, *Graphics Double Data Rate (GDDR5) SGRAM Standard*, Dec 2013.

[20] N. P. Jouppi, "Improving Direct-mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *ISCA*, Jun 1990.

[21] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *MICRO*, Dec 2011.

[22] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers." in *HPCA*, 2010.

[23] Y. Kim, M. Papamichael, O. Mutulu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, Dec 2010.

[24] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, Jun 2012.

[25] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, Feb 2013.

[26] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *ACM TACO*, vol. 10, no. 1, Apr 2013.

[27] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, Dec 1995.

[28] Micron Technology Inc., *Mobile LPDDR2 SDRAM Datasheet*, 2011.

[29] J. Mukundan and J. F. Martinez, "MORSE: Multi-Objective Reconfigurable Self-Optimizing Memory Scheduler," in *HPCA*, Feb 2012.

[30] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *ISCA*, Jun 2008.

[31] J. T. Pawlowski, "Hybrid Memory Cube," in *Hot Chips*, Aug 2011.

[32] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA*, Jun 2000.

[33] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, G. Marçais, M. Pop, and J. A. Yorke, "GAGE: A critical evaluation of genome assemblies and assembly algorithms," *Genome Research*, Dec 2011.

[34] Samsung Electronics, *DDR3 SDRAM Datasheet*, 2012.

[35] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, Oct 2002.

[36] A. Snavely and D. Tullsen, "Symbiotic Job Scheduling for a Simultaneous Mutlithreading Processor," in *ASPLOS*, Nov 2000.

[37] Y. H. Son, S. O, Y. Ro, J. W. Lee, and J. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, Jun 2013.

[38] T. Takahashi, T. Sekiguchi, R. Takemura, S. Narui, H. Fujisawa, S. Miyatake, M. Morino, K. Arai, S. Yamada, S. Shukuri, M. Nakamura, Y. Tadaki, K. Kajigaya, K. Kimura, and K. Itoh, "A Multi-gigabit DRAM Technology with 6F^2 Open-Bitline Cell, Distributed Overdriven Sensing, and Stacked-Flash Fuse," *Solid-State Circuits, IEEE Journal of*, vol. 36, no. 11, 2001.

[39] B. Wicht, S. Paul, and D. Schmitt-Landsiedel, "Analysis and Compensation of the Bitline Multiplexer in SRAM Current Sense Amplifiers," *Solid-State Circuits, IEEE Journal of*, vol. 36, no. 11, 2001.

[40] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *ISCA*, Jun 1995.

[41] W. Zhao and Y. Cao, "New Generation of Predictive Technology Model for Sub-45nm Design Exploration," in *ISQED*, Mar 2006.